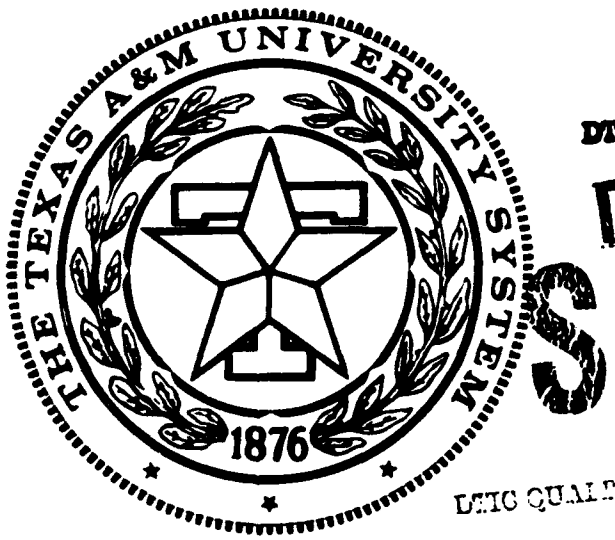
TEXAS A&M UNIVERSITY

Hardware Implementation of a Desktop Supercomputer for High Performance Image Processing

ONR Grant Number N00014-94-1-0516

Technical Report
Feb/01/94 – May/01/94



DEPARTMENT OF ELECTRICAL ENGINEERING
College Station, Texas

Hardware Implementation of a Desktop Supercomputer for High Performance Image Processing

ONR Grant Number N00014-94-1-0516

Technical Report
Feb/01/94 – May/01/94

CELLULAR NEURAL NETWORK BEHAVIORAL SIMULATION

Dr. Jose Pineda de Gyvez

Texas A&M University
Microelectronics Group
Department of Electrical Engineering
College Station, TX, 77843

Phone: (409) 8457477
FAX: (409) 8457161
Email:gyvez@pineda.tamu.edu

TABLE OF CONTENTS

1. INTRODUCTION

CNN is a hybrid of *Cellular Automata* and *Neural Networks* (hence the name Cellular Neural Networks), and it shares the best features of both worlds. Like Neural Networks, its continuous time feature allows real–time signal processing, and like Cellular Automata, its local interconnection feature makes VLSI realization feasible. Its grid–like structure is suitable for the solution of a high order system of first order non–linear differential equations on–line and in real–time . CNN is an analog nonlinear dynamic processor array, see Fig. 1a, characterized by the following features [2]:

1) Each analog processor is capable of processing continuous signals, in either continuos–time or discrete–time modes.
2) The processors are placed on a 3D geometric cellular grid (several 2D layers) and are basically identical.
3) Interaction among processors is local and mainly translation invariant.
4) The mode of operation may be transient, equilibrium, periodic, chaotic, or combined with logic (without A/D conversion).

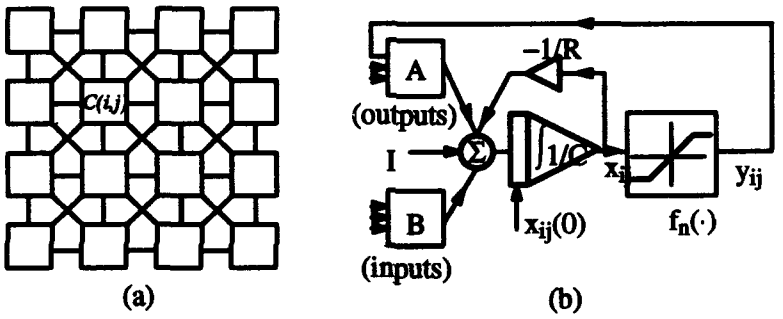


Fig. 1. CNN Structure and block diagram.

The basic circuit unit of CNN is called a *cell* [3]. It contains linear and nonlinear circuit elements. Any cell, *C(i,j)*, is connected only to its neighbor cells, i.e. adjacent cells interact directly with each other. This intuitive concept is called *neighborhood* and is denoted as *N(i,j)*. Cells not in the immediate neighborhood have indirect effect because of the propagation effects of the dynamics of the network. Each cell has a state x, input u, and output y. The state of each cell is bounded for all time $t > 0$ and, after the transient has settled down, a cellular neural network always approaches one of its stable equilibrium points. This last fact is relevant because it implies that the circuit will not oscillate. The dynamics of a CNN has both output feedback *(A)* and input control *(B)* mechanisms. The first order nonlinear differential equation defining the dynamics of a cellular neural network cell can be written as follows

$$C\frac{dx_{ij}(t)}{dt} = -\frac{1}{R}x_{ij}(t) + \sum_{C(k,l)\in N(i,j)} A(i,j;k,l)\ y_{kl}(t) + \sum_{C(k,l)\in N(i,j)} B(i,j;k,l)\ u_{kl} \qquad (1)$$

$$y_{ij}(t) = \frac{1}{2}(|x_{ij}(t) + 1| - |x_{ij}(t) - 1|)$$

where x_{ij} is the state of cell *C(i,j)*, $x_{ij}(0)$ is the initial condition of the cell, C is a linear capacitor, R is a linear resistor, I is an independent current source, $A(i,j;k,l)\ y_{kl}$ and $B(i,j;k,l)\ u_{kl}$ are voltage

controlled current sources for all cells $C(k,l)$ in the neighborhood $N(i,j)$ of cell $C(i,j)$, and y_{ij} represents the output equation.

Notice from the summation operators that each cell is affected by its neighbor cells. $A(\cdot)$ acts on the output of neighboring cells and is referred to as the *feedback operator*. $B(\cdot)$ in turn affects the input control and is referred to as the *control operator*. Specific entry values of matrices $A(\cdot)$ and $B(\cdot)$ are application dependent, are space invariant and are called *cloning templates*. A current bias I and the cloning templates determine the *transient behavior* of the cellular nonlinear network. The equivalent block diagram of a continuos–time cell implementation is shown in Fig. 1b.

CNNs have as input a set of analog values and its programmability is done via cloning templates. Thus, *programmability* is one of the most attractive properties of CNNs, but how to choose the optimal network and how to program it to perform a given task are still topics under investigation. This is the reason why there is a need for a behavioral CNN simulator capable of helping investigators design and manipulate cloning templates ("programming"). Existent tools are not meant to deal with a significant number of pixels typical in common image processing applications[6]. The simulator presented here not only satisfies this need, but it also can be used for *testing* CNN hardware implementations.

2. BEHAVIORAL SIMULATION

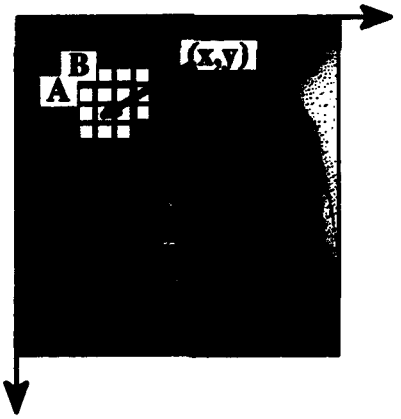


Fig. 2 Raster simulation approach

Recall that equation (1) is space invariant, which means that $A(i,j;k,l) = A(i–k,j–l)$ and $B(i,j;k,l) = B(i–k,j–l)$ for all i,j,k,l. Therefore, the solution of the system of difference equations can be seen as a convolution process between the image and the CNN processors. The basic approach is to imagine a square subimage area centered at (x,y), with the subimage being the same size of the templates involved in the simulation. The center of this subimage is then moved from pixel to pixel starting, say, at the top left corner and applying the A and B templates at each location (x,y) to solve the differential equation, see Fig. 2. This procedure is repeated for each time step, for all the pixels. An instance of this image scanning–processing is referred to as an "iteration". The processing stops when it is found that the states of all CNN processors have converged to steady–state values [3], and the outputs of its neighbor cells are saturated, e.g. they have a ± 1 value.

This whole simulating approach is referred to as *raster simulation.* A simplified algorithm is presented below for this approach. The part where the integration is involved (i.e. calculation of the next state) is explained in the Numerical Integration Methods section.

Algorithm: (Single–Layer or Raster CNN simulation)

Obtain the input image, initial conditions and templates from user;
/* M,N = # of rows/columns of the image */
while (converged_cells < total # of cells) {
  for (i=1; i<=M; i++)
    for (j=1; j<=N; j++) {
      if (convergence_flag[i][j])
      continue; /* current cell already converged */
      /* calculation of the next state*/

$$x_{ij}(t_{n+1}) = x_{ij}(t_n) + \int_{t_n}^{t_{n+1}} f'(x(t_n))\ dt$$

      /* convergence criteria */
      if ($\frac{dx_{ij}(t_n)}{dt} = 0$ and $y_{kl} = \pm 1,\ \forall\ C(k,l) \in N_r(i,j)$) {
      convergence_flag[i][j] = 1;
      converged_cells++ ;
      }
    } /* end for */
  /* update the state values of the whole image*/
  for (i=1; i<=M; i++)
    for (j=1; j<=N; j++) {
      if (convergence_flag[i][j]) continue;
      $x_{ij}(t_n) = x_{ij}(t_{n+1})$;
    }
  #_of_iteration++;
} /* end while */

The raster approach implies that each pixel is mapped onto a CNN processor. That is, we have an image processing function in the spatial domain that can be expressed as:

$$g(x,y) = T(f(x,y)) \tag{2}$$

where $f(\cdot)$ is the input image, $g(\cdot)$ the processed image, and T is an operator on $f(\cdot)$ defined over the neighborhood of (x,y). From hardware implementation's point of view, this is a very exhaustive approach. For practical applications, in the order of 250,000 pixels, the hardware would require an enormous amount of processors which would make its implementation unfeasible. An alternative is to multiplex the image processing operator. A *time–multiplexed* CNN simulator is presented in a companion paper [1].

3. NUMERICAL INTEGRATION METHODS

The CNN is described by a system of nonlinear differential equations. Therefore, it is necessary to discretize the differential equation for performing behavioral simulation. For computational purposes, a normalized time differential equation describing CNN is used [4]:

$$f'(x(n\tau)) = \frac{dx_{ij}(n\tau)}{d(n\tau)} = -x_{ij}(n\tau) + \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)\ y_{kl}(n\tau) + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)\ u_{kl} + I \tag{3}$$

$$y_{ij}(n\tau) = \frac{1}{2}(|x_{ij}(n\tau) + 1| - |x_{ij}(n\tau) - 1|)$$

where τ is the normalized time. For the purpose of solving the initial–value problem, well established Single–Step methods of numerical integration techniques are used [5]. These methods can be derived using the definition of the definite integral

$$x_{ij}((n+1)\tau) - x_{ij}(n\tau) = \int_{\tau_n}^{\tau_{n+1}} f'(x(n\tau))\ d(n\tau) \tag{4}$$

Three of the most widely used single–step algorithms are used in the CNN behavioral simulator discribed here. They are the Euler's algorithm, the Improved Euler Predictor–Corrector algorithm and the Fourth–Order (quartic) Runge–Kutta algorithm. These methods differ in the way they evaluate the integral presented in (4).

Euler's method is the simplest of all algorithms for solving ODEs. It is an explicit formula which uses the Taylor–series expansion to calculate the approximation

$$x_{ij}(n+1)\tau) = x_{ij}(n\tau) + \tau\ f'(x(n\tau)) \tag{5}$$

The Improved Euler Predictor–Corrector method uses both explicit (predictor) and implicit (corrector) formulae. The integral is calculated by multiplying the step size τ with the averaged sum of both the derivative of x(nτ) and the derivative of the predicted $x_p((n+1)\tau)$ at the next time step:

$$x_{ij}((n+1)\tau) = x_{ij}(n\tau) + \frac{\tau}{2}\ [f'(x(n\tau)) + f'(x_p((n+1)\tau)) \tag{6}$$

The Fourth–Order Runge–Kutta method is the most costly among the three methods in terms of computation time, as it requires four derivative evaluations per time step. However, its high cost is compensated by its accuracy in transient behavior analysis.

$$x_{ij}((n+1)\tau) = x_{ij}(n\tau) + \frac{k_1^{ij} + 2k_2^{ij} + 2k_3^{ij} + k_4^{ij}}{6} \tag{7}$$

where

$$k_1^{ij} = \tau\, f(x_{ij}(n\tau)) \qquad \forall_{C(l,m)\in N_r(i,j)}$$

$$k_2^{ij} = \tau\, f(x_{ij}(n\tau) + \frac{1}{2}k_1^{i,j}) \qquad \forall_{C(l,m)\in N_r(i,j)}$$

$$k_3^{ij} = \tau\, f(x_{ij}(n\tau) + \frac{1}{2}k_2^{i,j}) \qquad \forall_{C(l,m)\in N_r(i,j)}$$

$$k_4^{ij} = \tau\, f(x_{ij}(n\tau) + k_3^{i,j})$$

where $f(\cdot)$ is computed according to (1). There are many single–step methods available to us for this purpose. But, one option worth considering is the combination of two methods in solving for the solution. Since the fourth–order Runge–Kutta is among the most widely used single–step method for starting the solution of the initial–value problem in ODEs, the Predictor–Corrector method for continuing the solution can be combined with the Runge–Kutta starter to make a very efficient computer simulation method for solving the problem.

4. RASTER SIMULATION RESULTS AND COMPARISONS

All the simulations reported here are performed using a Sun SPARC2 workstation, and the simulation time used for comparisons is the actual CPU time used. The input image format for this simulator is the X windows bitmap format (xbm), which is commonly available and easily convertible from popular image formats like GIF or JPEG.

(a) (b)

Fig. 3. Image processing. (a) After Averaging Template
(b) After Averaging and Edge Detection Templates

Fig. 3 shows results of the raster simulator obtained from a complex image of 125,535 pixels. For this example an *Averaging* template followed by an *Edge Detection* template were applied to the original image to yield the images displayed in Figs. 3a and 3b, respectively.

Since speed is one of the main concerns in the simulation, finding the maximum step size that still yields convergence for a template can be helpful in speeding up the system. The speed–up can be achieved by selecting an appropriate Δt for that particular template. Even though the maximum step size may slightly vary from one image to another, the values in Fig.4 still serve as good references. These results were obtained by trial and error over more than 100 simulations on a diamond figure.

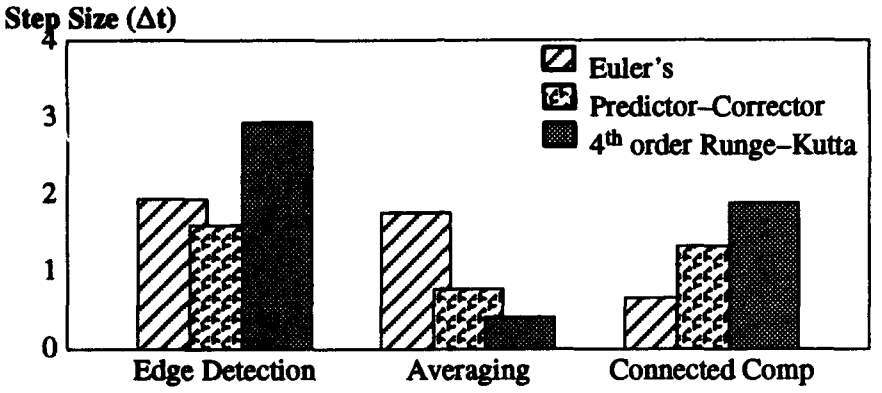


Fig. 4. Maximum step size that still yields convergence for 3 different templates

The importance of selecting an appropriate Δt can be easily visualized in Fig. 5. If the step size chosen is too small, it might take many iterations, hence longer time, to achieve convergence. On the other hand, if the step size taken is too large, it might not converge at all or it would converge to erroneous steady state values; the latter remark can be observed for the Euler integration method The results of Fig. 5 were obtained by simulating a small image of size 16x16 (256 pixels) using an Edge detection template on a diamond figure. In Fig. 6, simulation time computations using an Averaging template for images of sizes to about 250,000 pixels are shown.

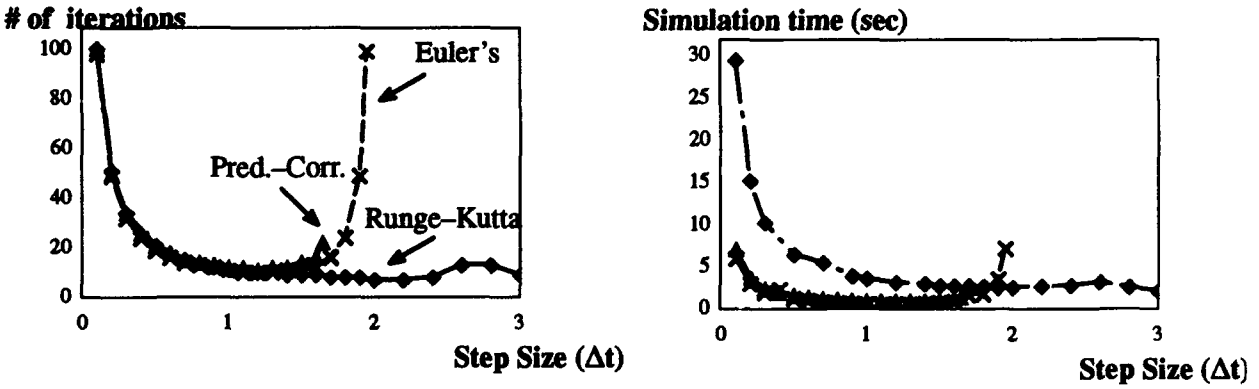


Fig. 5. Iteration & simulation time comparisons of the three methods using the Edge Detection template

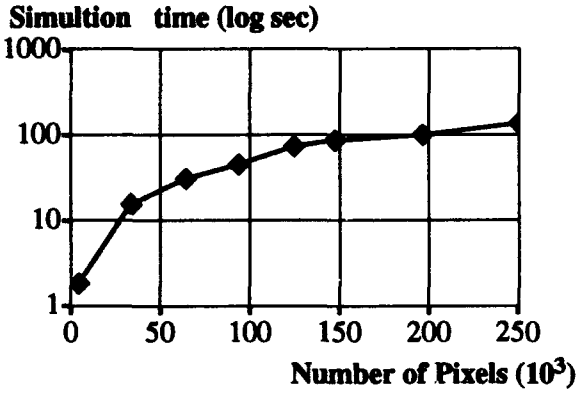


Fig. 6. Simulation time for images of sizes ranging from 64x64 (4096 pixels) to 415x603 (250245 pixels) using Averaging template

5. TIME MULTIPLEXING SIMULATION

Under this approach one can define a block of CNN processors which will process a subimage whose number of pixels is equal to the number of CNN processors in the block. The processing within this subimage follows the raster approach described in the companion paper [2]. Once convergence is achieved, a new subimage is processed. This procedure is repeated until the whole image has been scanned. It is obvious that with this approach the hardware implementation becomes feasible since now the number of CNN processors is finite. Also, the entire image is scanned only once since each block is allowed to fully converge.

Even though the approach seems tempting, an important observation is necessary: The processed border pixels in each subimage may have incorrect values since they are processed without neighboring information. Fortunately, the latency of CNNs is such that only local interactions are important. Hence, to cope with the previous problem, two sufficient conditions must be considered while doing time–multiplexing simulation. In other words, to ensure that each border cell properly interacts with its neighbors it is necessary: 1) to have a *belt of pixels* from the original image around the subimage, and 2) to have *pixel overlaps* between adjacent subimages.

It is possible to quantize the processing error of any border cell C_{ij} with neighborhood radius of 1. Let us compute independently the error due to the feedforward operator and then due to interactions among cells for two horizontally adjacent processing blocks. The absolute processing error due only to the effect of the B template is obtained by subtracting the erroneous state value from the error free states using eq 1. This yields,

$$\varepsilon_{ij}^{B} = \sum_{i=1}^{i=3} b_{i,j+1} \; sign(u_{i,j+1}) \qquad (8)$$

where $b_{i,j+1}$ are the missing entries from the B template due to the absence of input signals $u_{i,j+1}$ and $sign(\cdot)$ is the sign function. The latter function is used to represent the status of a pixel, e.g. *black* $\equiv 1$ and *white* $\equiv -1$. Notice that the error is both image and template dependent. In other words, the steady state of a border cell may converge to an incorrect value due to the absence of it's neighbors

weighted input. Given the local interconnectivity properties of CNN, one can conclude that the minimum width of the input belt of pixels is equal to the neighborhood radius of the CNN.

Let us study now the interactions among cells. For this effect, we can compute the absolute error in a similar form. Disregarding for the moment the B template this error is

$$\varepsilon_{ij}^{A} = \sum_{i=1}^{i=3} a_{i,j+1} \; y_{i,j+1}(t) \tag{9}$$

where $a_{i,j+1}$ are the missing entries from the A template due to the absence of weighted output signals $y_{i,j+1}$(t). The problem in this situation is more involved because the output signals depend on the state of their corresponding cells. To minimize the error an overlap of pixels between two adjacent blocks is proposed. The minimum overlap width must be proportional to 2× the neighborhood's radius of the CNN.

The general time–multiplexing procedure consists in iterating each block (subimage) until all CNN cells within the block converge. The block with converged cells will have state variables x which are the values used for the final output image. In the overlapping procedure the left side of the overlapped cells take converged values from $Block_i$ and the right side from $Block_{i+1}$, see Fig. 7. In our simulator the number of overlapping columns or rows between the adjacent blocks is defined by the user. Even though higher number of overlapping columns or rows means more accurate simulation of neighboring effects on the border cells, for applications where the correct final state is of more importance than the transient states, an overlap of two is usually sufficient. An even number overlapping of overlapping cells is recommended, since the converged cells in the overlapped region can be evenly divided by the two adjacent blocks.

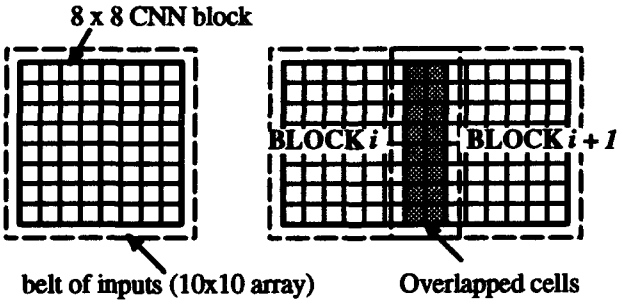


Fig. 7. CNN multiplexing with overlapped cells

With the added overlapping feature, better neighboring interactions are achieved, but at the same time, an increase in computation time is inevitable. However, by taking advantage of the fact that the original input image is been divided into small CNN subimages, the chance of a subimage having all its pixels black or white is high. This is another feature that can be added to the time–multiplexing simulation to improve computation times. The savings in simulation time come from avoiding repetitive simulations of all–black and all–white subimages.

The idea behind this time–saving scheme is that when the very first all–black/all–white block is encountered, after processing that block, the final states of the block are stored separately from the

whole image. When subsequent all–black/all–white blocks are found, there is no need to simulate these blocks since the converged states are readily available in memory, thus avoiding the most time consuming part of the simulation which is the numerical integration.

For the purpose of better understanding the overall idea of this simulation approach, the simplified algorithm is presented below:

Algorithm: (Time–Multiplexing CNN simulation)

$\mathfrak{B} = \{C_{ij} | i = 1,\ldots,block\_x \wedge j = 1,\ldots,block\_y\}$ $\mathcal{P} \subset \mathfrak{B}$ = *set of border cells (lower left corner)*
overlap = number of cell overlaps;
belt = width of input belt M = *number of rows of the image* N = *number of columns of the image*

```
for (i=0; i < M; i += block_x – overlap)
  for (j=0; j < N; j += block_y – overlap)
  {
    /* load initial conditions for the cells in the block except for those in the borders */
    for (p=–belt; p < block_x + belt; p++)
      for (q=–belt; q < block_y + belt; q++) {
```

$$x_{i+pj+j}(t_n) = \begin{cases} u_{ij} \\ 1 \quad \forall\ C_{i+pj+j} \in \mathfrak{B} \\ -1 \end{cases}$$

```
      } /* end for */
        /* if the block is all white or black don't
           process it */
      if(x_{i+p,j+q}=–1 ∨ x_{i+p,j+q}=1 ∀ C_{i+pj+q} ∈ 𝔅)
      {
          obtain the final states from memory;
          continue;
      }
    do { /* normal raster simulation */
      for (p=0; p < block_x; p++) {
        for (q=0; q < block_y; q++)
          { /* calculation of the next state excluding the belt of inputs */
```

$$x_{i+pj+q}(t_{n+1}) = x_{i+pj+q}(t_n) + \int_{t_n}^{t_{n+1}} f(x_{i+pj+q}(t_n))\, dt \qquad \forall\ C_{i+pj+q} \in \mathfrak{B}$$

```
            /* convergence criteria */
            if ( dx_{i+pj+q}(t_n)/dt = 0 and y_{kl} = ± 1 ∀ C(k,l) ∈ N_r(i + p, j + q) ) {
          converged_cells++ ;
        } /* end for */
          /* update state values */
      x_{i+pj+q}(t_n) = x_{i+pj+q}(t_{n+1}) ∀ C_{i+pj+q} ∈ 𝔅
    }
    } while( converged_cells < (block_x *block_y));
    /* store new state values excluding the ones corresponding to the border cells */
    𝒜 ← x_ij  ∀C_ij ∈ 𝔅 \ 𝒫
  } /* end for */
```

6. SIMULATION RESULTS AND COMPARISONS

The general features of the raster CNN simulator are preserved in the time–multiplexing simulator, namely the choiceof three integration methods, the format of the input image file and the capability of processing any size of input image. Some representative simulation results due to the effects of the added features in the time–multiplexing simulator are presented in this section.

In the time–multiplexing simulation involving the time–saving scheme, the number of all–black/all–white blocks that will be encountered during a simulation depends on the image itself and the block size chosen by the user. For example, Fig. 8a evidently will benefit from this time–saving scheme, especially from not needing to simulate more than once the all–white block.

Using actual numbers can easily show how much improvement is achieved. The size of Fig. 8a is 395x403 (159,185 pixels), and an edge detection template is used for simulation comparisons. First, using the raster simulator presented in [2], the simulation took 243.51 secs. Next, with the regular time–multiplexing simulator (with overlapping and input belt) the simulation took 363.28 secs. Finally, the time–multiplexing with the time–saving scheme performed the same simulation in 244.22 secs, almost a 33% improvement from the regular time–multiplexing. The size of the block used was 10x10, with two rows/columns overlapping.

Fig. 8. Time–multiplexing CNN simulation. (a) Original image. (b) After Edge Detection

By taking the lower right–hand corner of Fig. 8a, the cropped image in Fig. 9a, we can easily visualize the effects that the overlapping pixels and the belt of inputs have on the simulation. By choosing the block size to be 10x10, and applying an edge detection template, the results obtained by simulating the image without overlapping cells and belt of inputs and with the same features added, are shown in Fig. 9b and Fig. 9c, respectively. A clear lost of neighboring interaction is shown in Fig. 9b, which is recuperated by the overlapping and belt of inputs, shown in Fig. 9c.

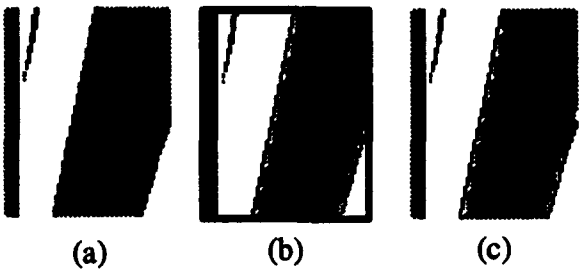


Fig. 3. Time–multiplexing simulation. (a) Original image. (b) *Without* overlapping & belt of inputs. (c) *With* overlapping of two & belt of inputs.

One interesting aspect to be considered from time multiplexing simulations is the optimum block size of the CNN. Fig. 10 displays timing simulation results versus different block sizes. Solid line results correspond to computations that were carried out on a 125,235 pixel–image without black/white blocks and using an Averaging Template. In this case, it can be observed that beyond a block size of 50x50 no noticeable CPU time improvement is made. The results presented by the dashed line correspond to computations that were carried out using an Edge Detection template on an image with many black/white blocks. For smaller block sizes many black/white blocks were found which makes the computation efficient. Beyond block sizes of 80x80 no more black/white blocks were found. This is why this is the peak computation time. Fig. 11 shows the average number of iterations within each CNN block and for the whole image for the Averaging Template case. These results show the complexity of operations that must be considered when deciding the actual CNN hardware size.

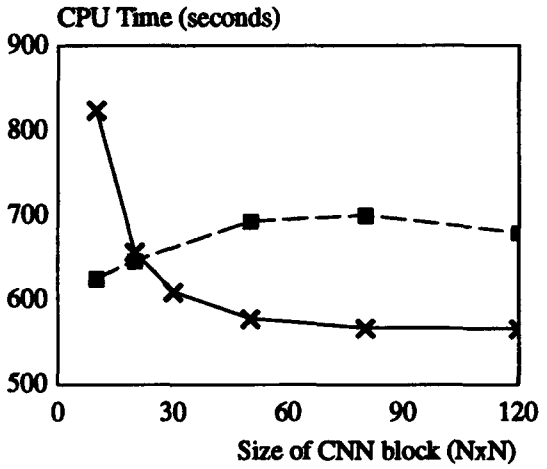


Fig. 10. CPU Time vs. CNN Block size

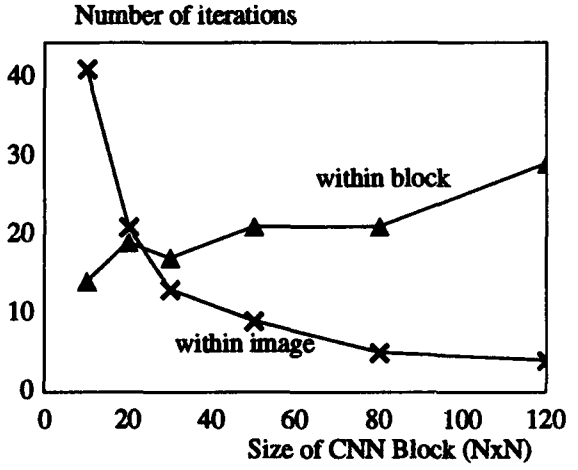


Fig. 11. Number of iterations vs. CNN Block Size

7. CONCLUSION

As researchers are coming up with more and more CNN applications, an efficient and powerful simulator is needed. The simulator hereby presented meets the need in three ways: 1) Depending on the accuracy required for the simulation, the user can choose from three popular methods to perform the numerical integration, 2) The input image format is the X Windows bitmap (xbm),

which is commonly available and 3) The input image can be of any size, allowing simulation of images available in common practices.

While keeping the features of the raster simulator, the time–multiplexing simulator presented here process the image block by block, simulating CNN the way the hardware would, if the number of CNN processors of the hardware is smaller than the input image, which usually it is the case with practical size images.

With the overlapping and external belt of inputs, the neighboring interaction between CNN blocks is ensured, but at the same time, computation costs also increased. However, with the added feature of processing the all–black and all–white blocks just once for the entire simulation, the simulation time is brought down to the levels of raster simulation, if not better, in some cases, depending on the input image and the size of block and overlap chosen.

8. REFERENCES

[1] C.C. Lee and Jose Pineda de Gyvez, "Time–Multiplexing CNN Simulator", *ISCAS'94*

[2] L.O. Chua and T. Roska, "The CNN Universal Machine Part 1: The Architecture", in *Int. Workshop on Cellular Neural Networks and their Applications (CNNA)*, pp. 1–10, 1992.

[3] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory & Applications," *IEEE Trans. Circuits and Systems*, Vol. CAS–35, pp. 1257–1290, 1988.

[4] J. A. Nossek, G. Seiler, T. Roska and L. O. Chua, "Cellular Neural Networks: Theory and Circuit Design," *International Journal of Circuit Theory and Applications*, Vol. 20, pp. 533–553, 1992.

[5] W. H. Press, B. P. Flannery, S.A. Teukolsky, and W.T. Vetterling, "Numerical Recipes. The Art of Scientific Computing", Cambridge University Press, New York, 1986

[6] J. Varrientos and E. Sanchez–Sinencio, "CELLSIM: A cellular neural network simulator for the personal computer," in Proc. 35th Midwest Symp. Circuits Systs, pp. 1384–1387, 1992.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 10, 1994	Technical Report 01/01/94-05/01/94

4. TITLE AND SUBTITLE
CELLULAR NEURAL NETWORK BEHAVIORAL SIMULATION

5. FUNDING NUMBERS
G
N00014-94-1-0516

6. AUTHOR(S)
Jose Pineda de Gyvez

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Texas Engineering Experiment Station
Texas A&M University

8. PERFORMING ORGANIZATION REPORT NUMBER
93-549/#1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Code 215: JWK
Ballston Tower One
800 North Quincy Street
Arlington, Virginia 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Results published in ISCAS '94

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

An efficient behavioral *simulator* for *Cellular Neural Networks* (CNN) is hereby reported. The simulator is capable of performing *Single–Layer* CNN simulations for any size of input image, thus a powerful tool for researchers investigating potential applications of CNN. This report presents an efficient *algorithm* exploiting the latency properties of Cellular Neural Networks along with numerical integration techniques; simulation results and comparisons are also presented. A novel approach to simulate the hardware of Cellular Neural Networks (CNN) is presented as well. The approach, *time–multiplexing* simulation, is prompted by the need to simulate hardware models and test hardware implementations of CNN. For practical size applications, due to hardware limitations, it is impossible to have a one–on–one mapping between the CNN hardware processors and all the pixels of the image. This simulator provides a solution by processing the input image block by block, with the number of pixels in a block being the same as the number of CNN processors in the hardware. The *algorithm* for implementing this simulator is also presented, along with some simulation results and comparisons.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Cellular, Neural Networks, Behavioral Simulating of CNN	13
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave Blank)

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency. Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ..., To be published in .... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availablity Statement. Denote public availability or limitation. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR)

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - DOD - Leave blank
DOE - DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports
NASA - NASA - Leave blank
NTIS - NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.